Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 63 (2015) 65 – 73

Procedia Computer Science

The 6th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2015)

A Domain Specific Language for High-Level Process Control Programming in Smart Buildings

Abdaladhem Albreshne[a*], Jacques Pasquier[a]

[a]*DIUF, University of Fribourg, 1700 Fribourg, Switzerland*

Abstract

Web services composition is a recurring idea in the field of smart residential environments (SRE), since it helps to solve complex problems such as energy saving, security control or health care by combining and orchestrating the available basic services. Smart environments are composed of networked devices (sensors and actuators) embedded within web services, which contain well-defined programming interfaces allowing them to share data, capture events and create composed control applications. There is still, however, a lack of domain specific languages (DSL) supporting a high degree of abstraction and transparency and allowing users to define control scenarios in a compact and comprehensible way. To satisfy these needs, the present paper aims to propose a DSL for describing scenarios to control SRE, with considerable gains in transparency, abstraction, expressiveness and simplicity.

Peer-review under responsibility of the Program Chairs

Keywords: Smart Residential Environments; Smart Objects; Sensors; Actuators; Web Services Composition and Orchestration; Ontology; Domain Specific Language.

1. Introduction

Recent projects and industry solutions in the domain of embedded devices have resulted in new communication protocols such as Zigbee[1] or 6LoWPAN[2], which allow the integration of low-powered devices with limited capacity into distributed computer networks. Additional middleware layers[3] based on Big web service (WS-*)[4] or RESTful web service[5] have been proposed to standardize the access to these devices and facilitate the discovery, data sharing

* Corresponding author. Tel.: +41-26-3008329; fax: +41-26-3009726
E-mail address: abdaladhem.albreshne@unifr.ch

Peer-review under responsibility of the Program Chairs
doi:10.1016/j.procs.2015.08.313

and events notifications. Thus, by enabling physical components to cooperate and share data, and with the help of appropriate software algorithms, it is possible to perform decision-making tasks, either autonomously or through human intervention, and thereby make the environment reactive and smart. The decision making process depends on collecting the necessary information about the environment and taking the appropriate actions.

In this context, it is essential to determine how the decision making task is defined. In other words, the question is how to enable the creation of comprehensive scenarios to control SRE on top of smart objects. To address this challenge, we proposed in [6,7] a generic software framework for managing SREs. It was designed to increase the transparency of accessing physical entities and to help integrate them into the application layer. To achieve this goal, the Ont4SRE[6,7] ontology and the GPL4SRE (Generic Process Language for SRE) domain specific language were proposed. The interested reader is invited to have a quick look at these papers in order to get a general understanding of the proposed framework architecture and of using the Ont4SRE ontology for discovering and integrating smart objects in complex scenarios. Indeed, the present paper focuses exclusively and in details on the GPL4SRE language, which is one of the fundamental parts of the proposed framework. In particular, it demonstrates how GPL4SRE enables the creation of complex control scenarios with a high degree of abstraction, transparency and expressivity.

The rest of this paper is organized as follows. Section 2 introduces a motivating scenario to serve as a guiding thread for the remaining sections. Section 3 briefly presents some related work. Section 4 gives an overview and explains, through the implementation of the motivating scenario, the benefits and features of the GPL4SRE language. Finally, Section 5 concludes the paper.

2. Motivating Scenario

In [6,7], we considered objects, appliances and locations as smart objects when they are coupled with miniaturized actuators and/or sensors connected to the network using the appropriate protocols. Web servers are embedded in these sensors and actuators or in smart proxies close to them, and implement web service technology like WS-* to expose services related to each smart object. On top of the service layer, an additional level of abstraction is introduced by marking up each service with semantic annotations based on the Ont4SRE ontology to describe its capabilities and thereby to facilitate its discovery and control process. In addition, a centralized discovery unit is available to assume responsibility for offering a query mechanism able to find relevant smart objects, the related services and their locations. Queries are formulated using the SPARQL[8] language.

Consider now a set of smart objects and appliances that occupy a given house, including smart rooms, heaters, shutters, showers and lamps. Listing 1 gives a high-level description (pseudo-code) of a control scenario example designed exclusively to be energy saving. Other examples could concentrate on comfort or security. Combining goals is also possible and has been tested but is outside the scope of this paper. To implement the energy saving scenario, the following challenges and requirements need to be explored:

1. ***Coordination***: the smart objects can be coordinated either through orchestration, where the smart objects are under the control of a single endpoint central unit called a process, or through choreography based on the cooperation between the distributed smart objects (participators). Each participator has to know exactly when to become active and with which components to interoperate. The question now is, which approach is suitable for the above scenario?
2. ***Capabilities***: whatever the coordination technique used, it needs a programming language to express its algorithms. This language should offer the following features:
    - *Interaction capabilities*: it should have the ability to interact with smart objects to acquire the current context of the environment (e.g. knowing the presence state of a given room (line 9)), react to new changes (e.g. being notified when the temperature changes (line 12)) and provide appropriate actions (e.g. switching all lamps off (lines 5, and 6)) in order to achieve the final goal, such as saving energy.
    - *Control flow*: it should offer control flow activities (`while`, `if`), which allow the creation of complex control scenarios. For example, check if the home is vacant (line 2) or perform a switch off action for each lamp (lines 5, 6).

- *Parallel and sequence execution*: when coordination is centralized, parallel execution is indispensable as the central process must react to events simultaneously over a long period. For example, different situations like in lines 1, 7, 15, 18 and 24 might be produced simultaneously.
- *Automatic discovery*: to force transparency, the discovery of smart objects and subscribing to events should be automatic. In lines 3-4, 5-6, we need to discover all the heaters and lamps in order to perform the desired actions. It is difficult to build control systems where all the involved services must be explicitly defined.
- *Handle Data*: it should be possible to use variables to store, reformat, manipulate and transfer messages.
- *Event handlers*: to allow the control system to respond to environmental changes by executing a specified set of operations independently. For example, switching a lamp off (line 11) depends on the presence detection (line 7), while switching a heater off (line 14) depends on temperature change (line 12). Providing an event handler mechanism makes the control system more aware of the context.
- *User interaction*: the end-user should be capable of interacting with the system to reconfigure and customize the control scenario according to his needs.

3. ***Abstraction***: there is a need to introduce the environmental conceptual model into the control programs by linking the domain knowledge to the source code. Therefore, the programmers write the control scenario in the same way as the environment is designed. This facilitates the writing, maintaining and understanding of control scenarios. These tasks would be much harder using a general-purpose language like BPEL4WS[9].
4. ***Transparency***: Any physical or software component should hide its implementation nature from its users, appearing and being used in a uniform way. Smart objects and their associated services should be clearly represented to the programmer. Users should declare them rather than specify where they are or how they are implemented.
5. ***Expressiveness***: the control scenarios should be designed and written in an understandable fashion, with fully meaningful phrases. This is to bridge the gap between natural human thinking and the syntax and expression of the language used. It should offer considerable gains in expressiveness and simplicity.
6. ***Reusability***: the language should be generic in order to support the creation of multi control scenarios in an SRE context such as saving energy, security etc.

```
1   When somebody leaves the home
2    if the home is vacant (homeState = vacant)
3     for each Heater
4      switchOff Heater
5     for each Lamp
6      switchOff Lamp
7   When somebody leaves a given room
8    wait for 3 minutes
9    query the presenceState in the Room
10   if nobody is in the room (presenceState = false)
11    switch off the Lamp in the given room
12  When a temperature has changed in a given room
13   if the temperature is greater than the maximum needed
14    switch off the Heater in the given room
15  When somebody starts taking a shower
16   wait 5 minutes
17   close the shower
18  When the day or the night starts
19   if it is day (daytime=true)
20    for each Shutter
21      open Shutter
22    for each lamp
23     switchOff lamp
24  When midnight is reached
25   for each Heater
26    adjust the Heater's targetTemperature to an economical level
```

Listing 1: Pseudo-code of the energy saving scenario

3. Related Work

At the application layer, numerous attempts related to the control of smart environments have been made. The entirely human-based programming approach is generally applied when the user has a well-defined control process model in mind. In that case, it is possible to use a general purpose process-oriented language such as BPEL4WS[9,10] or BPMN 2.0[11,12]. This kind of programming, however, forces users to make tedious efforts to find and understand detailed knowledge about services and environmental resources. The opposite approach consists of giving sufficient autonomy to control systems to let them take the appropriate decisions and to carry out actions. Such approaches are appropriate when the programmer does not have a comprehensive process, but rather, has a set of constraints and preferences. Several examples of this approach have been introduced in the literature[13], e.g., those based on Artificial intelligence planning[14], Linear-time Temporal Logic[15] (LTL) or Rule-Based planning[16]. Fully automatic control, however, is still viewed as a highly complex task because of the rapid proliferation of available services to choose from and the risks of drifting away from the initial user goal. The third approach can be qualified as semi-automatic. It finds and automatically integrates the appropriate smart objects, while the programmer defines the control workflow. Some efforts like in [17,18] apply semantic description and ontology to help improve both the discovery and composition operations. In this paper, the semi-automatic technique is adopted using a process-oriented DSL. The latter provides all the essential control and execution structures of a well-established process control language, such as BPEL4WS. In addition, it allows for transparently discovering and interacting with the smart objects. Other investigations in DSL languages for SRE can be found in [19,20].

4. The GPL4SRE Language

GPL4SRE is a workflow-based language proposed to orchestrate smart objects. It is based on BPEL4WS with the support of semantic declarations to simplify and make it more expressive. With this language, it is possible to discover and interact with different smart entities, create loops, declare variables, copy and assign values, as well as register and wait for events from sensor publishers. Scenario execution can be sequential or parallel. A GPL4SRE program has two main sections: a declaration and an execution one, as shown in Fig. 1.

1. **Declaration Section:** It includes the followings parts:
   - *Smart Entities Declaration Part* allows for the declaration of the smart entities that interact with the process. Smart entities can be declared individually or as a list of similar entities. Once a smart entity is declared, it can be controlled through its associated services. The smart entities are declared in regard only to their type and location (optional) as defined in the Ont4SRE ontology.
   - *Events Registration Part* allows subscribing the process to events that it is interesting in. The event subscription is performed with regard only to the event type as defined in the ontology (e.g. `MotionEvent`, `TemperatureChangingEvent`).
   - *Variables Declaration Part* allows the declaring of variables in order to receive manipulate and send data between the smart entities and the process. Variable can be global or local. The programmer is allowed to declare primitive data types such as `Int`, `Float`, and `Boolean`. It is possible also to declare complex variables in a `Map` collection form (i.e. a list of (name, value) pairs).
2. **Execution Section:** It specifies the parts that will be executed when the process is running. It includes the following parts:
   - *User Preferences Setting Part*: this plays the role of the initializer to customize the process according to the final user requirements. The process provides set construct to assign user preferences to variables, a `select` construct to select, according to the final user, which smart entity should be involved in the scenario, and a `print` construct to display information to the user through a simple interface.
   - *Main Thread Part* specifies the first part of the process that runs when it starts after the initialization of user preferences. This part can be used for the following proposals.
     - Initialize variables.
     - Perform a combination of actions at regular intervals (time-driven fashion).
     - Perform a combination of action when an event is produced (event-driven fashion), especially when a process reacts to an event over a short period of time.

- Determine the lifetime of the process. The process terminates when this part completes its execution.

- *Event / Alarm Handlers Part* are associated with an enclosed scope. This allows the process to respond to events or alarms (an expiration of a timeout) which occur independently of the execution of the main thread and are produced at any moment during the process's lifetime. The keyword `OnEvent` is used to declare the beginning of an event handler branch, and `OnAlarm` for the beginning of an alarm handler branch.

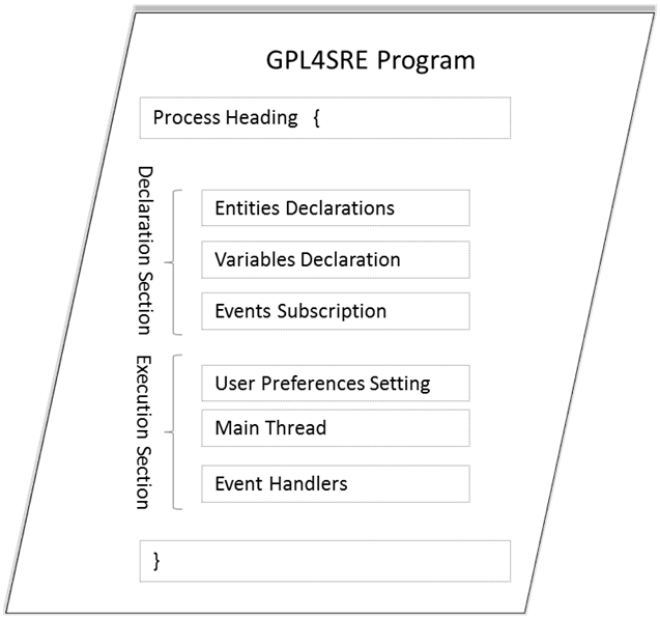

Fig. 1. The GPL4SRE structure

Now that we have seen the main structure of a GPL4SE program, we can present some of its principal constructs:

- *Control flow Constructs* define the program's procedural logic. The important ones are `for`, `while` and `if`.
- *Execution Constructs*: the workflow execution can be made sequential or parallel by using the keywords `sequence` and `flow`, respectively. Other execution constructs, inspired by BPEL4WS, are also provided, such as the `pick` construct to give a choice of possible sets of actions.
- *Service Interaction Constructs* allow for interacting with the smart objects to retrieve or change their current states. There are three types: (i) An `action` construct is used to invoke a service, which modifies the state of a given smart object; (ii) A `query` construct is used to invoke a service, which retrieves the current state; (iii) An `OnEvent` construct allows the process to be notified by an event publisher about any change in a particular smart object state. It is this set of constructs which constitutes the main strength of the proposed language compared to a general purpose one like BPEL4WS.

4.1. Scenario Implementation

Listing 2 shows how the previous energy saving scenario is written in GPL4SRE and the following points describe it:

- *Smart Entities Declarations*: lines 2-5 specify the smart objects involved. Smart objects can be declared individually or as a list of objects of the same type. The programmer specifies the object type, which is `Heater`, `Lamp` or `Room` according to their semantic definition in the Ont4SRE ontology.
- *Event Registration:* lines 6-7 register the process for a temperature change event and a presence detection event.
- *Variables Declarations:* line 9 declares a variable. In line 25, the `Map` data structure `input2` is declared. It holds the received message from the presence event.

- *User Preferences setting*: lines 10-17 set user preferences. The final user can express the temperature she considers hot and selects which heater can be involved in the control process.
- *The main Thread*: lines 18-24 specify the main body of the process.
- *Event Handlers:* lines 25, 35 and 41 specify the different event handlers executed in parallel. Decisions and actions are grouped according to these event handlers.
- *Control flow and Execution Constructs*: lines 18, 19, 21 and 22 give an example of the use of `sequence`, `while`, `if`, and `for` constructs, respectively.
- *Service Interaction Constructs*: line 22 shows how the `action` construct is used to invoke the `switchOff` service of a lamp and line 30 shows how the query construct is used to `query` the presence state of a room.

```
1   Process EnergySavingScenario {
2     List heaters  = discover(Heater);
3     List lamps    = discover(Lamp);
4     List rooms    = discover(Room);
5     Shower shower = discover(Shower,"shower_C");
6     subscribe TemperatureEvent;
7     subscribe AccessEvent;
8     ...
9     float hotTemp; int showerDuration;
10    preferences {
11    print("Which temperature do you consider hot in your home?");
12    set(hotTemp);
13    ...
14    for(Heater h: heaters){
15      print ("Would you like to use this heater?");
16      select (h);
17    }}
18    sequence{
19      while(true){
20        onEvent(AccessEvent, Map input1);
21        if(input1.get("homeState")== "vacant"){
22          for (Lamp lamp : lamps){action (lamp, "switchOff");}
23          for (Heater heater : heaters){action(heater,"switchOff");}
24    }}}
25    onEvent(PresenceEvent, Map input2){
26      String location = input2.get("locationID");
27      if (input2.get("presenceState") == false){
28       wait(3*60);
29       Room room = getEntityByLocation(rooms,location);
30       Map input3 = query(room,"getPresenceState");
31       if(input3.get("presenceState") == false ){
32         for(Lamp l: lamps){
33           if(l.getLocation().get("locationID") == location){action(l,"switchOff");
34    }}}}}
35    onEvent(TemperatureEvent, Map input5){
36      if(input1.get("temperature") > hotTemp){
37        for (Heater h: heaters){
38          if(h.getLocation().get("locationID") == location){
39            action(h,"switchOff");
40    }}}}
41    onEvent(ShowerEvent, Map input3){
42     if(input3.get("faucetState") == "opened"){
43       wait(showerDuration * 60);
44       action (shower, "close");
45    }}
46    onEvent(CalendarEvent, Map input4){...}
47    onAlarm(repeatEvery(23:59:59,0Y-0M-1D-0H-0M-1S)){...}
48  }
```

Listing 2: Code extract of the energy saving scenario

4.2. Evaluation

To validate the GPL4SRE language, the followings steps have been taken:

- A virtual (simulated) smart home with a large set of smart objects (including their associated actuators, sensors and event publishers) has been implemented.
- An ontology instance for the given home has been created.
- The registry & discovery engine was then applied to initialize the control process.
- Several scenarios, including the one presented in this paper, have been written with GPL4SRE and translated with the corresponding compiler into BPEL4WS executable programs.
- Finally, an execution environment based on the ORACLE SOA[21] platform has been used for executing and monitoring the scenarios.

To evaluate GPL4SRE, we discuss now its features with respect to the requirements introduced in Section 2:

1. ***Coordination***: There are advantages in using the orchestration approach as followed in this paper. Indeed, it is easier to design, comprehend and maintain without an advanced technical background. All the declarations are in one place and a scenario is nothing, but a story that everyone can comprehend. Furthermore, it is always possible to apply redundancy and fault-tolerant techniques to a central process. A decentralized approach might seem more robust, but it is also more complicated to tune-up, with the necessity to maintain potentially conflicting rules.
2. ***Capabilities****:* The implemented scenario revealed how GPL4SRE capabilities are designed to fit SRE requirements:
   - *Interaction capabilities*: As can be noted in lines 22, 30 and 44, GPL4SRE enables interactions with different smart objects, either in the form of queries or actions. This allows the control system to collect surrounding information from the environment in order to take suitable actions. Lines 37-40 illustrate such a group of actions within a `for` statement.
   - *Control flow*: As can be noted in lines 19 and 21, it is possible to use loops and conditional statements, thereby enabling the creation of complex control scenarios.
   - *Parallel and sequence execution*: Lines 18, 19 and 20 are executed sequentially, while the event handlers blocks (lines 25-34, 35-4, etc.) are executed in parallel to the main thread (lines 18-24).
   - *Automatic discovery and event registration*: Lines 2-7 show how the GPL4SRE enables the automatic discovery of smart objects, as well as the subscription to different events.
   - *Handle Data*: In lines 9 and 26 variables are declared to store and transfer messages.
   - *Event handler*: As noted in lines 25-34, 35-40 and 41-45, GPL4SRE allows differently produced events to be handled in parallel independently of, and asynchronously to, the execution of the main thread at any moment during the process' lifetime. This enables the control process to be informed about changes in the environment rather than having to frequently ask if any significant change has occurred.
   - *User interaction:* Lines 10-17 show how to involve the final user in the process through an ask-response dialogue.
3. ***Abstraction***: The GPL4SRE language enhances abstraction by introducing the environmental conceptual model of the Ont4SRE ontology into the source code. This helps developers to program in the same way as the environment is modeled and facilitates the writing, understanding and maintenance of the control scenarios, which would be harder using general-purpose languages like BPEL4WS. The abstractions carried out by the implemented scenario include:
   - *Smart objects declarations*: As illustrated in lines 2-7, the smart objects are declared according to their definition in the Ont4SRE ontology. The notions of `Heater`, `Room` and so on are used rather than syntax implantation notions like `partnerLink`.
   - *Services declarations*: In lines 30 and 39, the notions of `getPresenceState`, `switchOff` and so on, are used for the name of the invoked service, which corresponds to their definitions in the Ont4SRE ontology, rather than using their implemented syntax names.

- *Smart objects properties*: In lines 26-27, the smart objects properties (i.e. `presenceState`, `locationID`, etc.) are declared based on their definition in the ontology rather than by using classical WS-* inputs and outputs declarations.

4. ***Transparency***: The transparency discussed here means that any physical or software component should hide its implementation nature from its users, thus appearing and being used in a uniform and implicit way. The transparency carried out by the GF4SRE framework includes:
   - *Discovery and Registration transparency*: This includes hiding the implementation syntax and grounding details of each smart object and its associated services. As illustrated in lines 2-5, the `discover` method is used to find the different smart entities by specifying only their types (e.g. `Room`, `Heater`) and locations (optional), while discovering and linking the smart entities with their services is fully hidden, as the programmer should not have to be aware of that.
   - *Access and interaction transparency*: smart objects and their associated services are clearly represented to the programmer. Users declare them rather than specify where they are or how they are implemented. With a traditional composition language like BPEL4WS, the programmer needs to know a priori all the grounding details related to each service such as the service address, port type and operation implementation parameters (operation name, inputs, outputs).
5. ***Expressiveness***: With the GPL4SRE, there is continuity and consistency of the concepts, terms and expressions used. The relationships between the different concepts that form the SRE overall are clearly defined. This way of programming outlines the way in which a developer should program. A GPL4SRE programmer should keep in mind that there are smart objects that can be discovered, their actual states can be changed or read via `action` and `query` services and changes in their states can be detected using `onEvent` handlers. In the actual scenario, the smart entities (e.g. `Heater`, `Lamp`) keywords were used 19 times, the notion of `action` and `query` used 9 times and `discover` and `subscribe` keywords used 10 times. Further, each individual statement and code line has a meaning and is understandable by itself. Approximately 21% of the words and terms used in the scenario are Ontology-based keywords.
6. ***Reusability***: The attentive reader should note that, with the list discovery statement, the emergence of additional smart objects, similar to those already defined (e.g. heaters, lamps and rooms) will be automatically discovered and involved in the scenario without any extra programming efforts. Going further, the given scenario can be implemented in any home occupied by similar smart objects and services without configuring any device or providing any new implementation details.

Table 1 summaries the previous discussion in term of the characteristics and capabilities of GPL4SRE compared with a traditional composition language like BPEL4WS.

5. Conclusion

In this paper, we proposed a domain specific language that facilitates the definition and the execution of complex control scenarios in the context of smart residential environments. Smart objects are declared using ontology vocabulary to transparently discover them and their associated services (events, queries and actions). Secondly, the execution section offers, in a simplified manner all, the essential control structures of a well-established process control language such as BPEL4WS. The GPL4SRE language is designed to bridge the gap between natural human thinking and the syntax and expression of the language. It offers considerable gains in expressiveness and simplicity compared with other general-purpose language, where control scenarios are designed and written in understandable fashion with fully meaningful phrases using a precise vocabulary without having to program intricacies such as services grounding. We illustrated the features of the GPL4SRE language through an implemented scenario. We are actually actively exploring the challenges proposed by more complex scenarios, which might include conflicting goals such as optimizing energy, comfort and security simultaneously.

Table 1: A comparison of capabilities and syntax of GPL4SRE and BPEL4WS languages

Metric (18 smart objects are involved)	GPL4SRE scenario	BPEL4WS scenario
Automatic discovery	✓	×
Event registry	✓	✓ [a]
User preferences	✓	×
Variables declaration	✓	✓ [b]
Control flow and Execution Constructs	✓	✓ [b]
Smart objects declaration, services invocations and Event handlers	✓ (semantic-based)	✓ [a] (syntactic-based)
Iteration over a list of smart objects	✓	×
Code size	85 lines	484 lines
Language keywords	69	~ 605
Ontology-based keywords	21%	0%

✓[a] Programmer must manually bind each service and each event. ✓ [b] XML-based.

References

1. ZigBee Alliance. *ZigBee Specification*. www.zigbee.org [Accessed 19.05.2015].
2. Jonathan W. Hui, David E. Culler. *IP is dead, long live IP for wireless sensor networks.* In Proceedings of the 6th ACM Conference on Embedded network sensor systems. ACM; 2008. p.15-28.
3. Warriach Ullah Warriach. *State of the Art : Embedded Middleware Platform for A Smart Home*. International Journal of Smart Home, 2013; 7(6):275-294.
4. Nissanka Priyantha, Aman Kansal, Michel Goraczko, Feng Zhao. *Tiny web services: design and implementation of interoperable and evolvable sensor networks*. In: Proceedings of the 6th ACM conference on Embedded network sensor systems; ACM; 2008. p. 253-266
5. Mauro Caporuscio, M.F., Carlo Ghezzi, Valerie Issarny. *ubiREST: A RESTful Service-oriented Middleware for Ubiquitous Networking*. Advanced Web Services; Springer; 2014. p. 475-500.
6. Abdaladhem Albreshne, Ayoub Ait Lahcen, Jacques Pasquier. *A Framework and its Process Oriented Domain Specific Language for managing Smart Residential Environments*. In: International Journal of Smart Home; 2013; 7:377-392.
7. Abdaladhem Albreshne, Ayoub Ait Lahcen, Jacques Pasquier. *Using a Residential Environment Domain Ontology for Discovering and Integrating Smart Objects in Complex Scenarios*. In: Proceedings of the International Workshop on Enabling ICT for Smart Buildings (ICT-SB), Hasselt, Belgium. Elsevier; 2014; 32: 997-1002.
8. W3C. *SPARQL Query Language for RDF*. http://www.w3.org/TR/rdf-sparql-query/ [Accessed 20.05.2015].
9. OASIS. *Web Services Business Process Execution Language Version 2.0*. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html [Accessed 19.05.2015].
10. Nils Glombitza, Dennis Pfisterer, Stefan Fischer. *Integrating Wireless Sensor Networks into Web Service-Based Business Processes*. In: Proceedings of 4th International Workshop on Middleware Tools, and Run-Time Support for Sensor Networks. ACM; 2009.p. 25-30.
11. OMG. *Business Process Model and Notation. http://www.omg.org/spec/BPMN/2.0/* [Accessed 20.05.2015].
12. Feng Gao, Maciej Zaremba, Sami Bhiri, Wassim Derguerch. *Extending BPMN 2.0 with Sensor and Smart Device Business Functions*. In: Proceedings of 20th IEEE International Workshops on Enabling Technologies, Paris, France. IEEE; 2011.p. 297-302.
13. Thanos G. Stavropoulos, Dimitris Vrakas, Ioannis Vlahavas. *A survey of service composition in ambient intelligence environments*. Artificial Intelligence Review Journal, Springer, Netherlands ; 2013; 40(3): 247-270.
14. Eirini K., Ehasan W., Jaap B., Alexander L.. *Marco A. Interoperation, Composition and Simulation of Services at Home*. In: Processedings of 8th International Conference, ICSOC, Springer-Verlag, Berlin; 2010. p. 167-181.
15. De Masellis R., Di Ciccio C., Mecella M., Patrizi F. *Smart Home Planning Programs*. In: Processedings of 7th International Conference on Service Systems and Service Management, Tokyo; 2010. p. 1-6.
16. Manuel Garcia. *Easing the Smart Home: A rule-based language and multi-agent structure for end user development in Intelligent Environments*. Journal of Ambient Intelligence and Smart Environments ; 2010; 2(4):437-438
17. Feng Wang, Kenneth Turner. *An Ontology-based Actuator Discovery and Invocation Framework in Home Care Systems*. In: Processdings of 7th International Conference on Smart Homes and Health Telematics.Springer: Berlin; 2009. p. 66-73.
18. Katasonov A., Palviainen M. *Towards ontology-driven development of applications for smart environments*. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 8th IEEE International Conference: Mannheim; 2010. p.696-701.
19. Sanjin Sechic, Fei L., Schahram D. *COPAL: A Macro Language for Rapid Development of Context-aware Applications in Wireless Sensor Networks*. In: Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications, ACM; 2011. P. 1-6.
20. Adolf D., Ferranti E., Koch S. *SmartScript-A Domain-Specific Language for Appliance Control in Smart Grids*. In: Processedings of 3th International Conference On SmartGrid Communications, Tainan; 2012. p. 465-470.
21. ORACLE. *Oracle SOA Suite*. http://www.oracle.com/us/products/middleware/soa/suite/overview/index.html [Accessed 20/07/2015].